浙江大学实验报告

专业：计算机科学与技术
姓名：NoughtQ
学号：1145141919810
日期：2024 年 12 月 1 日

课程名称：图像信息处理　指导老师：宋明黎　成绩：＿＿＿

实验名称：图像双边滤波

一、 实验目的和要求

1. 暴力实现双边滤波

二、 实验内容和原理

2.1 双边滤波概述

双边滤波(bilateral filter)的目标是使图像更加平滑，具体来说：

- 保留大规模的特征——结构(structure)
- 去掉小规模的特征——纹路(texture)

它的大致思想是：

- 每张图像有两个主要特征：
  - 空间域S：在图像内所有可能位置的集合，与图像的分辨率相关（比如图像的行和列）
  - 强度域R：可能像素值的集合。对于不同的图像，用于表示像素值的位长可能因值的不同而变化，通常用无符号字节和浮点数来表示
- 每个样本点用它的相邻样本点的加权平均来代替
- 权重能够反映相邻样本点和中心样本点之间的接近和相似程度（因此更大的权重对应更接近、更相似的样本点）
- 所有的权重需要被归一化，以保留局部均值

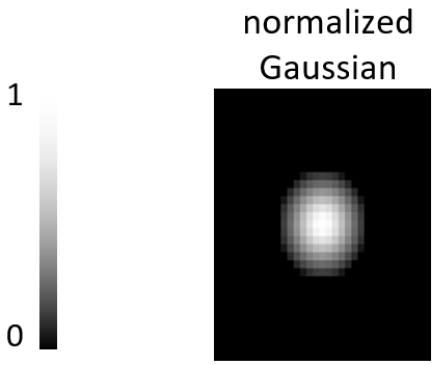
下面先来看一下双边滤波的一种简单情形——高斯滤波，之后我们会在高斯滤波的基础上实现双边滤波。

2.2 高斯滤波

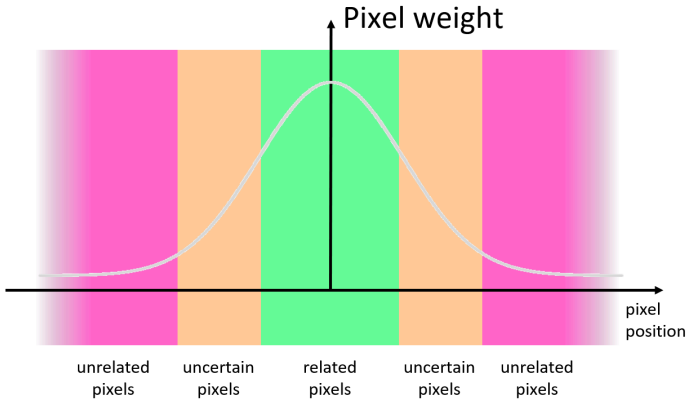
高斯滤波/模糊(Gaussian Filter/Blur)的公式如下所示，可以发现它是对像素的加权平均：

$$\mathrm{GB}[i]_p = \sum_{\boldsymbol{q}\in S} \underbrace{G_\sigma(|\boldsymbol{p}-\boldsymbol{q}|)}_{\text{normalized Gaussian}} I_q$$

用下括号标出的部分可以用灰度值表示：



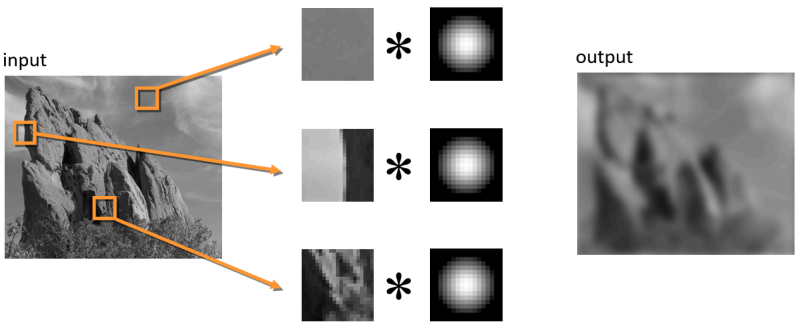
高斯函数$G_\sigma(x)$就是概统课上学的正态函数：$G_\sigma(x)=\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{x^2}{2\sigma^2}\right)$。高斯函数反映了：像素的权重根据其离中心点的位置成正态分布，即越靠近中间的像素点权重越大，表示是相关的像素点；离得越远就说明是不相关的像素点。



高斯函数中的参数σ将会影响高斯滤波的效果，因此需要根据实际情况选择合适的σ。通常可以采用以下策略：σ的值与图像大小呈正相关，比如令σ=图像对角线长的 2%，此时σ值与图像分辨率无关。

高斯滤波的性质：

- 能成功地平滑图像
- 但平滑过头了——它连图像内物体的边缘都给模糊掉了，因为它只考虑像素的空间距离，并没有考虑物体的边。对于不同的像素点，它可能采取相同的滤波方法，而没有考虑像素点的特征，因而把整张图都给模糊掉了。



2.3 双边滤波详述

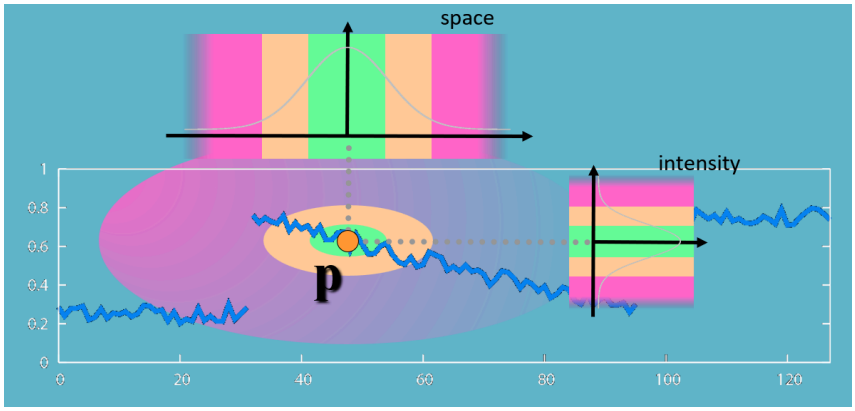
双边滤波克服了上述高斯滤波的缺陷——它在滤波的时候考虑到边的因素，公式如下所示：

$$\mathrm{BF}[I]_p = \frac{1}{W_p}\sum_{\boldsymbol{q}\in S}\underbrace{G_{\sigma_s}(|\boldsymbol{p}-\boldsymbol{q}|)}_{\text{Space weight}}\underbrace{G_{\sigma_r}(|I_p-I_q|)}_{\text{Intensity weight}}I_q$$

该公式在高斯滤波公式的基础上新增了：

- 归一化因数：$\frac{1}{W_p}=\sum_{\boldsymbol{q}\in S}G_{\sigma_s}(|\boldsymbol{p}-\boldsymbol{q}|)G_{\sigma_r}(|I_p-I_q|)$
- 强度权重，其中I_p表示中心点的像素值，I_q表示窗口内某一点的像素值

经过这番调整后，对于图像的每个像素点，我们只根据与该像素点在**空间上接近，强度上相似**的其他像素点来调整该像素点。如下图所示，绿色范围内的像素点便是能够影响像素点 P 的像素点。



双边滤波需要考虑 2 个参数：

- 空间参数σ_s：对应窗口大小，被考虑进来的像素点的空间范围
- 强度参数σ_r：对应边的明显程度

参数的确定还是取决于实际应用，比如：

- 空间参数：与图像大小成正比（前面已提到过）
- 强度参数：与边的明显程度成正比，比如图像梯度的平均数或中位数
- 参数应与图像的分辨率和曝光无关

我们可以对同一幅图像进行多次双边滤波，即迭代(iteration)，公式为：$I_{n+1}=\mathrm{BF}[I_n]$。这样可以形成一张按块光滑(piecewise-flat)的图像，但在计算机图像上通常不需要这种迭代。

> **双边滤波的缺陷**
>
> 由于双边滤波公式是非线性的，且窗口大小是复杂多变，不能提前确定的，因而双边滤波实际上很难计算，如果用暴力计算来实现的话相当慢。

上面的公式仅适用于灰度图，对于彩色图，只要稍微修改一下公式中的强度权重部分即可：

$$\mathrm{BF}[I]_p = \frac{1}{W_p}\sum_{\boldsymbol{q}\in S}\underbrace{G_{\sigma_s}(|\boldsymbol{p}-\boldsymbol{q}|)}_{\text{Space weight}}\underbrace{G_{\sigma_r}(|\boldsymbol{C_p}-\boldsymbol{C_q}|)}_{\text{Intensity weight}}\boldsymbol{C_q}$$

这里用$\boldsymbol{C_p}$和$\boldsymbol{C_q}$代替原来的I_p和I_q，它们是三维的向量，同时表示 RGB 三个通道的像素值。

应用：

- 去噪(denoising)
- 色调映射(tone mapping)
- 调整图像光强 & 纹理的编辑

三、 实验步骤与分析

3.1 双边滤波

先直接给出双边滤波操作相关的代码：

```
//  (不带系数的) 高斯函数
double gauss(double quad_x, double sigma) {
    return exp(- quad_x / (2 * pow(sigma, 2)));
}

// 双边滤波操作
BMPFILE BiFilter(BMPFILE bf) {
    BMPFILE newImg;
    BYTE val;
    LONG width, height;
    int x, y, i, j, tmpX, tmpY;
    int pos, halfLen;
    double curR, curG, curB;       // 中心像素的 RGB 值
    double sumR, sumG, sumB;
    double tmpR, tmpG, tmpB;
    double wR, wG, wB;             // 每个通道的归一化因数
    double gS, gR, gG, gB;         // 高斯函数的计算结果
    double sigma_s;                // 空间参数

    // 分配空间
    newImg = (BMPFILE)malloc(sizeof(struct tagBMPFILESTRUCT));
```

```
    // 将旧图的数据拷贝到新图上
    memcpy(&(newImg->bmfh), &(bf->bmfh), sizeof(BITMAPFILEHEADER));
    memcpy(&(newImg->bmih), &(bf->bmih), sizeof(BITMAPINFOHEADER));
    newImg->aBitmapBits =
            (BYTE *)malloc(sizeof(BYTE) * newImg->bmih.biSizeImage);

    width = newImg->bmih.biWidth;
    height = newImg->bmih.biHeight;
    halfLen = FILTERWINDOWLEN / 2;    // 窗口边长的一半
    // 空间参数 = 2% * 图像对角线的长度
    sigma_s = 0.02 * sqrt(pow(width, 2) + pow(height, 2));
     // 强度参数根据实际情况自行调节
    Sigma_r = 20;

    // 双边滤波
    for (y = 0; y < bf->bmih.biHeight; y++)
        for (x = 0; x < bf->bmih.biWidth; x++) {
            sumR = sumG = sumB = 0;
            wR = wG = wB = 0;
            pos = y * ColorBitWidth + x * 3;
            curB = bf->aBitmapBits[pos];
            curG = bf->aBitmapBits[pos + 1];
            curR = bf->aBitmapBits[pos + 2];
            for (i = -halfLen; i <= halfLen; i++)
                for (j = -halfLen; j <= halfLen; j++) {
                    // 越界处理
                    tmpY = y + i < 0 ? 0 :
                            (y + i >= height ? height - 1 : y + i);
                    tmpX = x + j < 0 ? 0 :
                            (x + j >= width ? width - 1 : x + j);
                    pos = tmpY * ColorBitWidth + tmpX * 3;
                    tmpB = bf->aBitmapBits[pos];
                    tmpG = bf->aBitmapBits[pos + 1];
                    tmpR = bf->aBitmapBits[pos + 2];
                    gS = gauss(pow(tmpY - y, 2) +
                               pow(tmpX - x, 2), sigma_s);
                    gR = gauss(pow(tmpR - curR, 2), Sigma_r);
                    gG = gauss(pow(tmpG - curG, 2), Sigma_r);
                    gB = gauss(pow(tmpB - curB, 2), Sigma_r);
                    // 归一化因数计算
                    wR += gS * gR; wG += gS * gG; wB += gS * gB;
                    sumR += gS * gR * tmpR;
```

```
                    sumG += gS * gG * tmpG;
                    sumB += gS * gB * tmpB;
                }
            // 更新中心像素的 RGB 值
            pos = y * ColorBitWidth + x * 3;
            newImg->aBitmapBits[pos] = rearrangeComp(sumB / wB);
            newImg->aBitmapBits[pos + 1] = rearrangeComp(sumG / wG);
            newImg->aBitmapBits[pos + 2] = rearrangeComp(sumR / wR);
        }

    return newImg;
}
```

这里对上述代码做一些必要的解释：

- 关于空间、强度参数：我根据课件上给出的建议，将图像对角线长度的 2% 作为空间参数值；但是对于强度参数，若按照课件建议的用梯度均值或中位数作为强度参数值，则有些过于复杂，因此就人为确定一个常数值作为强度参数值，根据滤波后的结果再做适当的调整。
- 关于滤波窗口的越界处理，我采用了与 Lab5 相同的策略，即对于超出范围的位置，我将其拉回到与该位置最接近的边界位置上。
- 关于高斯函数 `Gauss()`：可以看到这个函数没有乘上系数 $\frac{1}{\sqrt{2\pi}\sigma}$，这是因为双边滤波的计算中，分子和分母都用到了高斯函数，系数部分可以相互抵消，因此没有必要再乘上这个系数了。

3.2 主程序

主程序部分较为简单，故只给出代码而不作额外说明。

```
#include "bmp.h"
#include <stdio.h>
#include <stdlib.h>

int main() {
    BMPFILE oldImg, newImg;

    // 分配存储空间
    oldImg = (BMPFILE)malloc(sizeof(struct tagBMPFILESTRUCT));
    newImg = (BMPFILE)malloc(sizeof(struct tagBMPFILESTRUCT));

    // 读取原图
    oldImg = ReadBMPFile();

    // 进行双边滤波处理
```

```
    newImg = BiFilter(oldImg);

    // 得到新图
   printf("Now the program is generating the bilateral-filterd image,
wait a minute...\n");
    WriteBMPFile(newImg, 17);

    return 0;
}
```

四、 实验环境及运行方法

4.1 实验环境

我的开发环境如下:

- Windows 11 24H2
- gcc version 14.2.0
- GNU Make 4.4.1

4.2 运行方法

若要运行主程序，需要遵循以下步骤:

> **注意**
>
> 在执行以下命令前，请检查一下路径下 `./代码/test` 是否存在`.bmp` 图片，并且检查一下`./代码/scripts/bmp.h` 文件的宏定义 `BMPFILEPATH` 是否指的是该图片。若有问题请自行修改，否则程序无法正常运行。

1. 将目录切换至 `./代码/build`
2. 执行以下命令:

```
# 编译代码
$ make

# 运行可执行文件
$ ./lab6
Successfully open the file!
Size: 1548022(bit)
ColorBitWidth: 2144
Width: 714
Height: 722
```

```
Image Size: 1547968
Now the program is generating the bilateral-filterd image, wait
a minute...
Finish the conversion successfully!
```

3. 来到 `./代码/tests` 目录，此时可以看到双边滤波后的图像。

五、 实验结果展示

> **注意**
>
> 本报告采用 typst 书写，但是 typst 不支持插入 BMP 图像文件，因此这里展示的是它们的 PNG 形式，对应的 BMP 形式见目录 `./代码/tests`。

先比对不同强度参数下的双边滤波效果：

Figure 5: 原图

Figure 6: $\sigma_r = 20$

Figure 7: $\sigma_r = 50$

Figure 8: $\sigma_r = 100$

可以看到，如果强度参数 σ_r 过大，会使原图像变得更加模糊一些，因此我就选取 $\sigma_r = 20$ 作为之后双边滤波的参数值。

再展示不同滤波窗口下的双边滤波效果：

Figure 9: 窗口大小 5×5

Figure 10: 窗口大小 9×9

Figure 11: 窗口大小 15×15

可以看到，窗口越大，图像的噪点更少，更加平滑些，即滤波效果更好。将双边滤波的结果与上一次实验中的均值滤波进行比对，显然双边滤波的效果会更好些。

这里没有再使用更大的窗口，这是因为窗口过大导致程序运行时间过长，故没有往后尝试。下面再试一张从课件上摘取下来的图片（$\sigma_r = 20$）：

Figure 12: 原图

Figure 13: 窗口大小 5×5

Figure 14: 窗口大小 15×15

Figure 15: 窗口大小 25×25

六、 心得体会

这次实验相对来说也是比较轻松的，因为只需要完成一项任务，且有了上次实现均值滤波和拉普拉斯变换的锐化滤波的经验（处理窗口越界问题等），因此本次实验的完成还算较为顺利的。相比均值滤波呈现的效果，这次双边滤波的效果相当不错，能够去掉图像的噪点且保持原图像的质量，感觉很像某些美颜相机上用到的操作，还蛮有趣的。总之这次实验让我对双边滤波的知识有了更深的理解，使我受益良多。